Pós-Graduação Lato Sensu

Curso de Especialização em Tecnologia Java

APOSTILA – IMPLEMENTANDO RELACIONAMENTOS BIDIRECIONAIS COM JPA

Professor Hugo Baker Goveia

hbakergoveia@hotmail.com

UTFPR

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CORNÉLIO PROCÓPIO

1. Relacionamentos bidirecionais com JPA

Relacionamentos bidirecionais são aqueles em que duas entidades estão associadas entre si, permitindo a navegação e o acesso de ambos os lados da associação. Por exemplo, se você tem uma entidade **Categoria** e uma entidade **Música**, um relacionamento bidirecional permitiria que uma **Categoria** saiba quais **Músicas** ela esta relacionada e uma Música saiba qual é a sua Categoria diretamente ao acessar a entidade, sem necessidade de fazer uma consulta extra para saber.
A JPA é uma especificação que define como os objetos Java são mapeados para as tabelas de um banco de dados relacional, e ela oferece suporte a relacionamentos bidirecionais por meio de anotações e configurações.

Pontos Positivos dos Relacionamentos Bidirecionais:

Navegação Conveniente: Relacionamentos bidirecionais facilitam a navegação entre as entidades associadas. Você pode acessar informações do lado oposto da associação sem precisar de uma nova consulta ao banco de dados. A JPA permite que você acesse entidades associadas de ambos os lados do relacionamento, facilitando a navegação e o acesso aos dados.

Eficiência em Consultas: Relacionamentos bidirecionais podem melhorar a eficiência em consultas complexas que envolvem várias entidades. Isso pode reduzir a necessidade de consultas separadas. Pois você pode acessar informações do lado oposto da associação sem emitir novas consultas ao banco de dados.

Integridade do Modelo: Em muitos casos, um relacionamento bidirecional reflete mais fielmente a natureza das relações no mundo real entre as entidades.

Flexibilidade em Operações: Relacionamentos bidirecionais permitem que você gerencie as operações de associação e desassociação de maneira mais eficiente, pois ambos os lados da associação são conhecidos.

Pontos Negativos dos Relacionamentos Bidirecionais:

Complexidade: Relacionamentos bidirecionais podem adicionar complexidade ao código e ao modelo de dados. Gerenciar as duas direções do relacionamento requer atenção extra.

Potencial para Erros: Se não forem gerenciados adequadamente, os relacionamentos bidirecionais podem levar a erros sutis, como loops infinitos ao imprimir entidades ou problemas de sincronização.

Custo de Desempenho: Relacionamentos bidirecionais podem aumentar o número de operações de banco de dados necessárias para manter a consistência dos relacionamentos. Isso pode afetar o desempenho, especialmente em operações em lote.

Manutenção: Mudanças em um lado do relacionamento podem exigir atualizações no outro lado. Isso pode aumentar a complexidade das mudanças e a manutenção do código.

Conclusão
A decisão de usar relacionamentos bidirecionais depende das necessidades específicas do projeto. Eles são úteis quando a navegação de ambos os lados do relacionamento é frequentemente necessária e quando faz sentido refletir a associação bidirecional do mundo real no modelo de dados. No entanto, em alguns casos, **relacionamentos unidirecionais** podem ser mais simples e eficientes.
É importante compreender as vantagens e desvantagens de relacionamentos bidirecionais ao projetar seu modelo de dados e considerar cuidadosamente o impacto que eles podem ter na manutenção, desempenho e complexidade.

2. Implementação usando MappedBy

Vamos considerar o projeto implementado como exemplo em nossas apostilas do curso e atualizar ele para contemplar relacionamentos bidirecionais entre as entidades.

Para isso usaremos a anotação mappedBy
A anotação mappedBy é usada em JPA (Java Persistence API) para estabelecer um relacionamento bidirecional entre duas entidades. Ela indica que o lado oposto da associação (ou seja, o lado "dono" do relacionamento) já está mapeado por outra entidade e, portanto, não precisa ser mapeado novamente na classe atual.

O uso do mappedBy tem alguns propósitos importantes:

Evitar Ambiguidade: Quando você tem um relacionamento bidirecional, ambos os lados do relacionamento possuem chaves estrangeiras para o outro lado. Se você não usar o mappedBy, o JPA tentará criar duas colunas para representar o mesmo relacionamento no banco de dados, o que pode levar a duplicações e ambiguidades.

Economia de Recursos: Usar o mappedBy evita que você precise duplicar as configurações de mapeamento em ambos os lados do relacionamento. Isso torna o código mais limpo e reduz a possibilidade de erros de configuração.

Consistência: Ao usar o mappedBy, você garante que ambos os lados do relacionamento permaneçam sincronizados. Se você atualizar um lado do relacionamento, o outro lado será automaticamente atualizado no banco de dados.

Melhoria de Desempenho: Usar o mappedBy pode melhorar o desempenho, pois reduz a quantidade de trabalho que o JPA precisa fazer para manter os relacionamentos.

Para implementar:

1. Na classe entidade Música a única alteração que devemos fazer é sobrescrever o método **toString()** para evitar que ao realizar consultas envolvendo essa entidade entremos em loops infinitos. Pois em relacionamentos bidirecionais, se você não

sobrescrever o toString(), pode ocorrer uma recursão infinita, já que uma entidade pode chamar o toString() da outra e vice-versa. Sobrescrevendo o método, você pode controlar o que é exibido.

Musica.java

```java
package com.utfpr.backendcategoriamusicasi.entity;

import jakarta.persistence.*;
import lombok.Data;

6 usages
@Entity
@Table(name = "musica")
@Data
public class Musica {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "cod_musica", nullable = false)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "cod_categoria", nullable = false)
    private Categoria categoria;

    private Integer duracao;

    @Column(length = 100)
    private String titulo;

    @Override
    public String toString() {
        return "Musica{" +
                "id=" + id +
                ", categoria=" + categoria.getDescCategoria() +
                ", duracao=" + duracao +
                ", titulo='" + titulo + '\'' +
                '}';
    }
}

```

2. Adicionar o atributo musicas na classe entidade Categoria, com o mappedBy informado na anotação, as importações corretas e sobrescrever o **toString()**

Categoria.java

```java
package com.utfpr.backendcategoriamusicasi.entity;

import jakarta.persistence.*;
import lombok.Data;

import java.util.List;

7 usages
@Entity
@Table(name = "categoria")
@Data
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "cod_categoria", nullable = false)
    private Long id;

    @Column(name = "desc_categoria", length = 50)
    private String descCategoria;

    @OneToMany(mappedBy = "categoria", fetch = FetchType.EAGER)
    private List<Musica> musicas;

    @Override
    public String toString() {
        return "Categoria{" +
                "id=" + id +
                ", descCategoria='" + descCategoria + '\'' +
                ", musicas=" + musicas +
                '}';
    }
}

```

É importante notar que na anotação **@OneToMany** foi incluído o **fetch = FetchType.EAGER.** Foi implementado dessa forma para evitar que ocorra erros quando o Hibernate tenta buscar informações sobre uma coleção (nesse caso, a lista "**músicas**" na classe "**categoria**" e não consegue fazer isso porque a sessão do Hibernate foi fechada.

Nesse sentido foi aplicado o **Eager Loading** que faz as coleções serem carregadas imediatamente junto com a entidade pai.

Porém é importante ressaltar que o contexto de uso sempre deve ser analisado e decidido de acordo com o tipo do projeto, pois pode não ser eficiente em todos os casos, especialmente se você tiver muitos registros na coleção.

3. Ao observar o console, podemos ver que foi impresso a lista de músicas dentro do toString() de Categoria.

```
============Listagem de todas as CATEGORIAS
Categoria{id=1, descCategoria='MPB', musicas=[Musica{id=1, categoria=MPB, duracao=240, titulo='Amor I love you'}, Musica{id=2, categoria=MPB, duracao=300, titulo='Nao e faci
Categoria{id=2, descCategoria='Rock', musicas=[Musica{id=4, categoria=Rock, duracao=500, titulo='Daniel na cova dos leões'}, Musica{id=5, categoria=Rock, duracao=322, titulo=
Categoria{id=3, descCategoria='Vira', musicas=[Musica{id=8, categoria=Vira, duracao=298, titulo='Vira-vira'}]}
Categoria{id=4, descCategoria='Bossa Nova', musicas=[Musica{id=9, categoria=Bossa Nova, duracao=348, titulo='Chega de saudade'}, Musica{id=10, categoria=Bossa Nova, duracao=2
Categoria{id=5, descCategoria='Jazz', musicas=[Musica{id=19, categoria=Jazz, duracao=446, titulo='New York'}]}
Categoria{id=6, descCategoria='Pop rock', musicas=[Musica{id=13, categoria=Pop rock, duracao=333, titulo='Politik'}, Musica{id=14, categoria=Pop rock, duracao=225, titulo='Gr
```